On the Security of Modular Exponentiation with Application to the Construction of Pseudorandom Generators*

Oded Goldreich
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science
Rehovot, Israel.
oded@wisdom.weizmann.ac.il

Vered Rosen
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science
Rehovot, Israel.
veredr@wisdom.weizmann.ac.il

December 5, 2000

Abstract

Assuming the inractability of factoring, we show that the output of the exponentiation modulo a composite function $f_{N,g}(x) = g^x \bmod N$ (where $N = P \cdot Q$) is pseudorandom, even when its input is restricted to be half the size. This result is equivalent to the simultaneous hardness of the upper half of the bits of $f_{N,g}$, proven by Håstad, Schrift and Shamir. Yet, we supply a different proof that is significantly simpler than the original one. In addition, we suggest a pseudorandom generator which is more efficient than all previously known factoring based pseudorandom generators.

Keywords: Modular exponentiation, discrete logarithm, hard core predicates, simultaneous security, pseudorandom generator, factoring assumption.

*This write-up is based on the Master Thesis of the second author (supervised by the first author).

1 Introduction

One-way functions play an extremely important role in modern cryptography. Loosely speaking, these are functions which are easy to evaluate but hard to invert. A number theoretic function which is widely believed to be one-way, is the exponentiation function over a finite field. Its inverse, the discrete logarithm function, is the basis for numerous cryptographic applications. Most applications use a field of prime cardinality, though many of them can be adapted to work in other algebraic structures as well.

A concept tightly connected to one-way functions is the notion of *hard-core predicates*, introduced by Blum and Micali [BM]. A polynomial-time predicate b is called a hard-core of a function f, if all efficient algorithm, given $f(x)$, can guess $b(x)$ with success probability only negligibly better than half. Blum and Micali showed the importance of hard-core predicates in pseudorandom bit generation. Specifically, they showed that the modular exponentiation function over a field of prime cardinality, $f_{P,g}(x) = g^x \bmod P$, has a hard-core predicate, and used it in order to construct a pseudorandom bit generator. The study of hard-core predicates of $f_{P,g}$ has culminated in the work of Håstad and Näslund [HN], showing that all bits of $f_{P,g}$ are individually secure.

1.1 Hard core functions

The concept of a hard-core function (or the simultaneous security of bits) is a generalization of hard-core predicates. Intuitively, a sequence of bits associated to a one-way function f is said to be simultaneously secure, if no efficient algorithm can gain any information about the given sequence of bits in x, given only $f(x)$. Proving the simultaneous security of a sequence of bits (rather than a single bit) in $f_{P,g}$ is a desirable result, enabling the construction of more efficient pseudorandom generators as well as improving other applications. However, the best known result regarding the simultaneous security of bits in $f_{P,g}$ is due to Long and Wigderson [LW], Kalisky [Kal] and Peralta [P], who showed that $O(\log n)$ bits are simultaneously secure, where n is the size of the modulus P.

Stronger results were demonstrated when the modulus was taken to be a composite, thus allowing to relate (simultaneous) hardness of bits to the factoring problem. Denote by $f_{N,g}$ the exponentiation modulo a composite function, defined as $f_{N,g}(x) = g^x \bmod N$, where N is an n-bit composite equal to the multiplication of two large primes and g is an element in the multiplicative group mod N. Håstad, Schrift and Shamir showed that under the factoring intractability assumption, all the bits in $f_{N,g}$ are individually hard, and that the upper $\lceil \frac{n}{2} \rceil$ bits and lower $\lceil \frac{n}{2} \rceil$ bits are simultaneously hard [HSS].

In the same setting (and under the same assumption that factoring is hard), we show that no efficient algorithm can tell apart $f_{N,g}(r)$ from $f_{N,g}(R)$, where r is a random $\lceil \frac{n}{2} \rceil$-bit string and R is a random n-bit string [1]. That is, one can work with an exponent x of half the size, and still obtain an element which "seems random" to all efficient algorithms. Note that all the cryptographic tools that use exponentiation in Z_N^* (and base their security on the discrete logarithm assumption) can greatly benefit from this fact, since the time consumed for exponentiation grows linearly with the size of the exponent (and is thus cut by a factor of two). Our result is in fact equivalent to the result of Håstad et.al. [HSS] on the simultaneous hardness of the upper $\lceil \frac{n}{2} \rceil$ bits of $f_{N,g}$. Nevertheless, we give an alternative proof for it while using some of their ideas and techniques. Our approach significantly simplifies the proof given in [HSS] and sheds a new light on it.

[1] As a matter of fact, in the exact formulation of our result, R is uniformly distributed over the range of naturals smaller than the order of g (in the group Z_N^*). However, the above claim (with R uniformly distributed in $\{0,1\}^n$) holds as well, as an implication of Lemma 3.3.

Another implication of our work (to be further discussed below) is the construction of a pseudorandom bit generator based on the computational indistinguishability of $f_{N,g}(r)$ from $f_{N,g}(R)$. Our generator is somewhat more efficient than all previously known factoring based pseudorandom generators.

1.2 An efficient Pseudorandom generator

The notion of a pseudorandom bit generator, introduced by Blum and Micali [BM], plays a central role in cryptography. It enables the user to expand a short random seed into a longer sequence of bits, that can be used in any efficient application instead of a truly random bit sequence. Blum and Micali presented a pseudorandom bit generator based on the discrete log problem. Using the fact that the exponentiation function over a field of prime cardinality has a hard-core predicate, they suggested an iterative generator that yields one bit of output per each exponentiation. Furthermore, they conceived a general paradigm that constructs an iterative pseudorandom generator, given any length preserving one-way permutation f, and a hard-core predicate b for f.

The Blum-Blum-Shub pseudorandom generator [BBS], hereafter referred to as the "BBS generator", is based on the above paradigm, taking f to be the modular squaring function, where the modulus N is a Blum integer.[2] Since, as shown by Rabin [R1], the problem of factoring N can be reduced to the problem of extracting square roots in the multiplicative group mod N, the function f is a one-way function assuming the intractability of factoring Blum integers. Additionally, Blum, Blum and Shub showed that f induces a permutation over the set of quadratic residues in the multiplicative group mod N, and using the results of Alexi et.al. [ACGS] and Vazirani and Vazirani [VV], this implies that the least significant bit constitutes a hard-core predicate for f. The BBS generator is by far more efficient than the Blum-Micali generator.[3] In particular, for every polynomial $P(\cdot)$, the BBS generator stretches an n-bit seed into a $P(n)$-bit pseudorandom string using $P(n)$ modular multiplications.

Another generator whose pseudorandomness is based on factoring, was suggested by Håstad, Schrift and Shamir [HSS] (and will be referred to as the "HSS generator"). The HSS generator relies on the simultaneous hardness of half of the bits in the exponentiation modulo a composite function $f_{N,g}$. Loosely speaking, the HSS generator takes an n-bit random seed x (where n is the size of the modulus N), and outputs $f_{N,g}(x)$ followed by the lower half of the bits of x.[4] Observe that from an n-bit seed, the HSS generator obtains $1.5n$ bits of output, using n modular multiplications on the worst case, and $0.5n$ modular multiplications on the average case (assuming that the terms $g^{2^0}, \ldots, g^{2^n}$ are pre-computed together with the other parameters of the generator).

Even though our main result is equivalent to the simultaneous hardness of half of the bits in $f_{N,g}$, our result gives rise to a pseudorandom generator that is (in a sense) more natural than the HSS generator, as well as more efficient than it. Informally, we suggest a generator that takes a random seed x of size $\lceil n/2 \rceil$, and outputs $f_{N,g}(x)$. Observe that our generator doubles the length of its input. In particular, it obtains n bits of output from an $0.5n$-bit seed using $0.5n$ modular multiplications on the worst case, and $0.25n$ modular multiplications on the average case (once again, we assume that the terms $g^{2^0}, \ldots, g^{2^{\lceil n/2 \rceil}}$ are pre-computed).

[2] A Blum integer is equal to the multiplication of two primes of equal size, each congruent to 3 mod 4.

[3] The Blum-Micali generator obtains each bit of output at the cost of one modular exponentiation that is implemented by n modular multiplications, as opposed to one modular multiplication per output bit needed by the BBS generator.

[4] As a matter of fact, in order to achieve true pseudorandomness, universal hashing is applied. The actual construction will be presented in Section 4.

The following table compares the three factoring based generators discussed above, each having the same security parameter n (the size of the modulus N). Note that the "cost" column refers to the average number of multiplications done in every application of the generator, and the "amortized cost" column refers to the average number of multiplications divided by the number of additional output bits of the generator (i.e., the amortized cost is the cost divided by the difference between the output length and the seed length).[5]

	seed length	**output length**	**cost**	**amortized cost**
BBS construction	n	P(n) $\quad (\forall P)$	P(n)	$\frac{P(n)}{P(n)-n} \approx 1$
HSS construction	n	1.5n	0.5n	$\frac{0.5n}{1.5n-n} = 1$
Our construction	0.5n	n	0.25n	$\frac{0.25n}{n-0.5n} = 0.5$

An additional point is that our generator (as well as the HSS generator) has an efficient parallel implementation in time $O(\log n)$. [6] This is opposed to the BBS generator which is not known to have a fast parallel implementation (i.e., any faster than the straightforward sequential implementation).

1.3 Organization

The rest of this work is organized as follows: Basic definitions and notations are given in Section 2. In Section 3 we state and prove the main theorem (regarding the pseudorandomness of exponentiation with a short exponent), show its equivalence to the [HSS] result (and discuss the difference between the two proofs). In Section 4 we address the issue of constructing a pseudorandom generator based on results as ours and [HSS].

2 Preliminaries

Probability Ensembles: Let I be a countable index set. A **probability ensemble indexed by** I is a sequence of random variables indexed by I. Namely, $X = \{X_i\}_{i \in I}$, where the X_i's are random variables, is a probability ensemble indexed by I.

In our applications, we use $\mathbb{N}$ as an index set, and let each X_n (in an ensemble of the form $\{X_n\}_{n\in\mathbb{N}}$) range over strings of length n. In particular, we denote by U_n the random variable that is uniformly distributed over $\{0,1\}^n$.

Statistical Difference: A basic notion from probability theory is the *statistical difference* between probability ensembles $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$. The statistical difference measures the dis-

[5]Even though the correct way to compare the above generators is with respect to the same security parameter, one might consider a comparison with respect to the same seed length. In order to do that we must normalize the input/output sizes of our generator so that its seed length will be n. Thus, the output produced by our generator will be of length $2n$, the cost will be $0.5n$ and the amortized cost will again be 0.5 multiplications per an additional output bit. Note however, that the size of the security parameter in our construction will be twice its size in the BBS and the HSS constructions. Thus, our construction will be safer. On the other hand, each multiplication will involve twice as big numbers.

[6]The parallel implementation uses $\lceil n/2 \rceil$ processors $P_1, \ldots, P_{\lceil n/2 \rceil}$, where the input of each processor P_i is the i'th bit of the seed, s_i, and the output is the multiplication of the values $g^{2^{i-1} \cdot s_i}$ contributed by each processor.

tance between distributions and is defined to be

$$SD(X_n, Y_n) = \frac{1}{2} \cdot \sum_{\alpha} |\Pr[X_n = \alpha] - \Pr[Y_n = \alpha]|$$

Probability ensembles $\{X_n\}_{n \in \mathbb{N}}$ and $\{Y_n\}_{n \in \mathbb{N}}$ are called **statistically close** if their statistical difference is *negligible* in n (we say that a function $\mu : \mathbb{N} \to [0,1]$ is **negligible** if for every positive constant c and all sufficiently large n's, $\mu(n) < \frac{1}{n^c}$).

Computational Indistinguishability: A weaker notion of closeness between probability ensembles is the notion of indistinguishability by all efficient algorithms. When no efficient algorithm (that may be probabilistic) can tell apart the two ensembles, we call them computationally indistinguishable. Formally,

Definition 2.1 *We say that two ensembles $\{X_n\}_{n \in N}$ and $\{Y_n\}_{n \in N}$ are* **computationally indistinguishable***, if for every probabilistic polynomial-time algorithm D, for every positive constant c and for all sufficiently large n's*

$$|\Pr[D(X_n, 1^n) = 1] - \Pr[D(Y_n, 1^n) = 1]| < \frac{1}{n^c}$$

A notation: Let A be a finite set, then $a \in_R A$ denotes that the element a is uniformly chosen from the set A (i.e. with probability $\frac{1}{|A|}$).

2.1 Pseudorandom Generators

Loosely speaking, a pseudorandom generator is a deterministic algorithm that stretches a random *seed* (i.e. input) into a longer bit sequence which is "pseudorandom". A pseudorandom bit sequence is defined as computationally indistinguishable from the uniform distribution (thus for all practical purposes we can use the output of the generator instead of a truly random string).

Definition 2.2 *A* **pseudorandom generator** *is a deterministic polynomial-time algorithm, G, satisfying the following two conditions:*

1. *There exists a function $l(n) : n \to n$ satisfying that $l(n) > n$ for all $n \in N$, such that $|G(s)| = l(|s|)$ for all $s \in \{0,1\}^*$.*

2. *The ensembles $\{G(U_n)\}_{n \in N}$ and $\{U_{l(n)}\}_{n \in N}$ are computationally indistinguishable.*

2.2 The Factoring Assumption

We denote by N_n the set of all n-bit integers $N = P \cdot Q$, where P and Q are two odd primes of equal size. The collection N_n can be sampled efficiently. Specifically, given input 1^n, it is possible to pick a random element in N_n in polynomial time (using a polynomial number of coin tosses).

The problem of factoring integers is widely believed to be intractable. Integers belonging to the set N_n are considered to be particularly hard to factor. Note that N_n is a non-negligible fraction of all n-bit integers. Currently, the best algorithm known can factor an integer picked randomly from N_n in (heuristic) running-time of $e^{1.92 n^{1/3} \log n^{2/3}}$.

Assumption 1 *[Factoring Assumption] Let A be a probabilistic polynomial-time algorithm. There is no constant $c > 0$ such that for all sufficiently large n's*

$$Pr\left[A(P \cdot Q) = P\right] > \frac{1}{n^c}$$

where $N = P \cdot Q$ is picked uniformly from N_n.

2.3 The group Z_N^*

Denote by Z_N^* the multiplicative group that consists of all the naturals which are smaller than N and are relatively prime to it. We represent the elements in Z_N^* by binary strings of size $n = \lceil \log N \rceil$.

Notations:

- Let $x < N$, and let $1 \le j \le i \le n$. We denote by x_i the i'th bit in the binary representation of x, and by $x_{i,j}$ the substring of x including the bits from position j to position i.
- Denote by $ord_N(g)$ the order of an element g in Z_N^*, which is the minimal $k \ge 1$ for which $g^k = 1 \pmod N$.
- Denote by $\langle g \rangle$ the subgroup of Z_N^* generated by g. That is, $\langle g \rangle$ is the set of all elements of the form $g^x \bmod N$ for some $x < N$.
- Denote by P_n the set of pairs $\langle N, g \rangle$ where $N \in N_n$ and $g \in Z_N^*$. Note that P_n is efficiently samplable.

We now define the *exponentiation modulo a composite* function and its inverse the *discrete logarithm modulo a composite* function.

Definition 2.3 *Let $\langle N, g \rangle$ be a pair in P_n. We define he exponentiation modulo a composite function $f_{N,g} : \{0,1\}^* \to \langle g \rangle$ to be $f_{N,g}(x) = g^x \bmod N$.*

Definition 2.4 *Let $\langle N, g \rangle$ be a pair in P_n. We define the discrete log modulo a composite function $DL_{N,g} : \langle g \rangle \to [0, ord_N(g))$, where $DL_{N,g}(y)$ is defined to be the unique natural $x < ord_N(g)$ for which $f_{N,g}(x) = y$.*

3 Exponentiation with a short exponent is pseudorandom

We introduce two probability ensembles, which we show to be computationally indistinguishable assuming the intractability of factoring.

Definition 3.1 *Let $\langle N, g \rangle$ be a uniformly distributed pair in P_n, let R be uniformly distributed in $[0, ord_N(g))$ and let r be uniformly distributed in $\{0,1\}^{\lceil \frac{n}{2} \rceil}$. We denote by $Full_n$ the distribution $\langle N, g, g^R \bmod N \rangle$ and by $Half_n$ the distribution $\langle N, g, g^r \bmod N \rangle$.*

Theorem 3.2 *The ensembles $\{Half_n\}_{n \in N}$ and $\{Full_n\}_{n \in N}$ are computationally indistinguishable.*

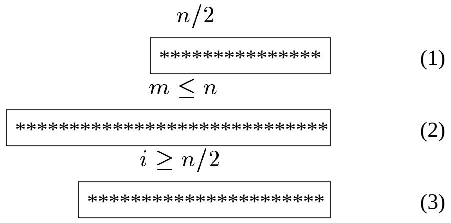

Figure 1: We denote random bits by '*' and the length of the binary expansion of $ord_N(g)$ by m. No. (1), (2), (3) show the exponents of $Half_n$, $Full_n$ and the hybrid H_n^i, respectively.

We use the hybrid technique in order to prove the indistinguishability of $Full_n$ and $Half_n$. For i's between $\lceil \frac{n}{2} \rceil$ and $n + \omega(\log n)$ we define a hybrid distribution in the following way: The i'th hybrid, denoted H_n^i, will consist of triplets of the form $\langle N, g, g^x \bmod N \rangle$, where $\langle N, g \rangle$ is uniformly distributed in P_n and x is uniformly distributed in $\{0,1\}^i$ (see Figure 1).

For a specific choice of a pair $\langle N, g \rangle$ in P_n we denote by $H_{N,g}^i$ the distribution $g^x \bmod N$ where x is uniformly distributed in $\{0,1\}^i$. (From now on we omit the expression "mod N" whenever it is clear from the context).

Clearly, $H_n^{\lceil n/2 \rceil} = Half_n$. Note that the distribution $H_n^{n+\omega(\log n)}$ is statistically close to $Full_n$, as asserted by the following claim:

Claim 3.2.1 *The distributions $Full_n$ and $H_n^{n+\omega(\log n)}$ are statistically close.*

Proof: Let M denote $2^{n+\omega(\log n)}$. M can be written as $k \cdot ord_N(g) + r$ where k is an integer and $0 \leq r < ord_N(g)$. We now calculate the statistical difference between the distributions $Full_n$ and $H_n^{n+\omega(\log n)}$. Note that the first equality is implied from the fact that in $f_{N,g}(x)$ the exponent x is reduced modulo $ord_N(g)$.

$$\begin{aligned}
SD(Full_n, H_n^{n+\omega(\log n)}) &= \frac{1}{2}\left[r \cdot \left(\frac{k+1}{M} - \frac{1}{ord_N(g)}\right) + (ord_N(g) - r) \cdot \left(\frac{1}{ord_n(g)} - \frac{k}{M}\right)\right] \\
&= \frac{1}{2}\left[(ord_N(g) - 2r) \cdot \left(\frac{1}{ord_n(g)} - \frac{k}{M}\right) + \frac{r}{M}\right] \\
&= \frac{1}{2}\left[(ord_N(g) - 2r) \cdot \frac{r}{M \cdot ord_N(g)} + \frac{r}{M}\right] \\
&\leq \frac{r}{M}
\end{aligned}$$

Since $\frac{r}{M} < \frac{N}{M} \leq \frac{2^n}{2^{n+\omega(\log n)}}$, we have that $SD(Full_n, H_n^{n+\omega(\log n)})$ is negligible in n. $\square$

Consequently, if there exists a probabilistic polynomial-time algorithm D, that distinguishes the ensemble $Half_n$ from $Full_n$, then D distinguishes (almost) as well $Half_n$ from $H_n^{n+\omega(\log n)}$. As the total number of hybrids is polynomial in n, a non-negligible gap between the extreme hybrids translates into a non-negligible gap between a pair of neighboring hybrids. Taking advantage of the structure of two neighboring hybrids, we use the distinguisher D in order to factor a composite in N_n, and thus contradict Assumption 1. In the following, let n be a sufficiently large natural and let i belong to the set $\{\lceil \frac{n}{2} \rceil, \ldots, n+\omega(\log n)\}$.

Lemma 3.3 (Main Lemma) *Suppose that the gap between the acceptance probability of D on the hybrids H_n^i and H_n^{i+1} is greater than $\frac{1}{n^c}$. Then, with probability at least $\frac{1}{8n^c}$ we can factor a composite N, uniformly distributed in N_n.*

3.1 Factoring vs Discrete Logarithm in Z_N^*

It turns out that there is a tight connection between factoring N and revealing the discrete logarithm of a certain element in Z_N^*. In order to factor a random integer $N = P \cdot Q$ in N_n, it is sufficient to find the discrete log of g^N for a randomly chosen $g \in Z_N^*$. This is due to the following trivial fact:

Fact 1 *Let $\langle N, g\rangle$ belong to P_n (say that $N = P \cdot Q$). Then, if $ord_N(g) > P + Q - 1$, the discrete logarithm $S = DL_{N,g}(g^N)$ is equal to $P + Q - 1$.*

Proof: Recall that the order of g divides the order of the group Z_N^*, equal to $\varphi(N) = (P-1)(Q-1)$. Therefore, $g^N = g^{N-\varphi(N)} = g^{P+Q-1} \pmod N$. Consequently, if $ord_N(g) > P + Q - 1$ then $S = P + Q - 1$. □

The following proposition, established by Håstad et al. [HSS], claims that an element picked randomly in Z_N^* is very likely to be of high order:

Proposition 3.4 (Håstad et al.) *Let $\langle N, g\rangle$ be uniformly distributed in P_n, where $N = P \cdot Q$. Then,*

$$\Pr\left[ord_N(g) < \frac{1}{n^k} \cdot (P-1)(Q-1)\right] \leq O\left(\frac{1}{n^{(k-4)/3}}\right)$$

The only use we make of the above proposition, is to show that with very high probability, $ord_N(g)$ cannot be too small. Specifically, Proposition 3.4 implies that with overwhelming probability $ord_N(g)$ is greater than $P + Q - 1$. Therefore, as was first observed by Chor [Chor], we can solve the two equations $P + Q - 1 = S$ (according to Fact 1) and $P \cdot Q = N$ for the unknowns P and Q and thus factor N.

3.2 Proof of Main Lemma

The proof of Lemma 3.3 is basically a reduction. We show how to use the algorithm D that distinguishes H_n^i and H_n^{i+1} in order to calculate S and thus factor N.

3.2.1 Using D to discover the $(i+1)^{st}$ bit of the exponent

Let $W_n \subseteq P_n$ be the set of pairs $\langle N, g\rangle$ in P_n for which it holds that D distinguishes $H_{N,g}^i$ and $H_{N,g}^{i+1}$ with advantage at least $\frac{1}{2n^c}$. A standard averaging argument shows that the probability that a pair $\langle N, g\rangle$ chosen at random from P_n is in the set W_n is at least $\frac{1}{2n^c}$.
¿From now on we consider the case where $\langle N, g\rangle$ belongs to the set W_n, and therefore satisfies

$$\left|\Pr[D(N,g,g^x) = 1 | x \in_R \{0,1\}^i] - \Pr[D(N,g,g^x) = 1 | x \in_R \{0,1\}^{i+1}]\right| \geq \frac{1}{2n^c} \tag{1}$$

Observe that

$$\begin{aligned}\Pr[D(N,g,g^x) = 1 | x \in_R \{0,1\}^{i+1}] = \ & \tfrac{1}{2} \cdot \Pr[D(N,g,g^x) = 1 | x \in_R \{0,1\}^i] \ + \\ & \tfrac{1}{2} \cdot \Pr[D(N,g,g^{2^i+x}) = 1 | x \in_R \{0,1\}^i]\end{aligned} \tag{2}$$

¿From 1 and 2 we obtain the following:

$$\left|\Pr[D(N,g,g^x) = 1 | x \in_R \{0,1\}^i] - \Pr[D(N,g,g^{2^i+x}) = 1 | x \in_R \{0,1\}^i]\right| \geq \frac{1}{n^c} \tag{3}$$

Denote by $\overline{H}^i_{N,g}$ the distribution g^{2^i+x} where x is drawn uniformly from $\{0,1\}^i$. Another way to state Inequality 3 is to say that the distinguisher D has advantage at least $\frac{1}{n^c}$ in distinguishing the distributions $H^i_{N,g}$ and $\overline{H}^i_{N,g}$. Let β and γ be the acceptance probabilities of D on input taken from $H^i_{N,g}$ and $\overline{H}^i_{N,g}$, respectively. That is, let

$$\beta \stackrel{def}{=} \Pr[D(N,g,g^x) = 1 | x \in_R \{0,1\}^i] \tag{4}$$

and

$$\gamma \stackrel{def}{=} \Pr[D(N,g,g^{2^i+x}) = 1 | x \in_R \{0,1\}^i] \tag{5}$$

Without loss of generality assume that $\gamma > \beta$. Note that good approximations of β and γ can be easily obtained (in polynomial-time) by performing a-priori tests on D, using samples taken from $H^i_{N,g}$ and $\overline{H}^i_{N,g}$.

In the sequel we use the distinguisher D as an oracle, that enables us to "peek" into a 1-bit window on the $(i+1)^{st}$ location of an unknown exponent of length $(i+1)$. Specifically, we use D in order to derive the $(i+1)^{st}$ bit of an $(i+1)$-bit string x, given g^x.

3.2.2 Discovering S - a naive implementation

Suppose for a moment that we had a "perfect" oracle, that given input $Z = g^x$, where x is of length $(i+1)$, would supply us, with success probability 1, the $(i+1)^{st}$ bit of x. It would then enable us to extract x, using two simple operations:

Shifting to the left: By squaring Z we shift x by one position to the left.

Zeroing the j'th bit: By dividing Z by $g^{2^{j-1}}$ we zero the j'th position in x, in case it is known to be 1.

Therefore, we extract x from the most significant to the least significant bit by "moving" it under the $(i+1)^{st}$ window. Specifically, we query the oracle and determine the $(i+1)^{st}$ bit of x and zero it in case it equals 1. Next we shift x by one position to the left, query again the oracle to discover the next bit and so on.

As was explained earlier, we try to factor N by discovering $S = DL_{N,g}(g^N)$. An important property of S is that with overwhelming probability its length is $\lceil n/2 \rceil + 1$, and is therefore smaller than $i+1$. We can thus manipulate $Y = g^N = g^S \pmod N$ and discover S.

However, as the oracle might give us erroneous answers and all we are guaranteed is that there is a $\gamma - \beta$ gap (which is greater than $\frac{1}{n^c}$) between the probability to get a correct 1-answer and the probability to get an erroneous 1-answer, our implementation needs to be more careful.

3.2.3 Discovering S - the actual implementation

We must randomize our queries to the oracle and learn the correct answer by comparing the proportion of 1-answers with β and γ. A straightforward way to learn the $(i+1)^{st}$ bit of x given Z, would be to query the oracle on polynomially many random multiples $Z \cdot g^{r_k}$ for known r_k's chosen uniformly from $\{0,1\}^i$, and based on the fraction of 1-answers to decide between 0 and 1. However this approach fails, since despite our knowledge of r_k, we cannot tell whether a carry from the addition of the i least significant bits of the known r_k and the unknown x effects the $(i+1)^{st}$ bit of their sum. Thus we cannot gain any information on the $(i+1)^{st}$ bit of x from the answer of the oracle on $Z \cdot g^{r_k}$.

We now give a rough description of a procedure that resolves this difficulty and computes S. The procedure consists of $\lceil n/2 \rceil + 1$ stages, where on the j'th stage we create a list L_j which is a subset of $\{0, \ldots, 2^j - 1\}$. We want two invariants to hold for the list L_j:

1. L_j contains an element e such that $S - e \cdot 2^{l(j)}$ belongs to the set $\{0,\ldots,2^{l(j)}-1\}$, where $l(j) \stackrel{def}{=} \lceil \frac{n}{2} \rceil + 1 - j$. (In other words, we want e to be equal to $S_{\lceil \frac{n}{2} \rceil + 1, l(j)}$).
2. The size of L_j is small, that is, it contains up to a polynomial number of values (where the polynomial is set a-priori).

Thus, on the $(\lceil n/2 \rceil + 1)^{st}$ stage, we will have a polynomial-size list that contains S.

The values in each list are kept sorted. The transition from the $(j-1)^{st}$ list to the j^{th} list is done as follows: We first let L_j contain all the values v such that $v = 2u$ or $v = 2u+1$ where u is in L_{j-1}, thus making the size of L_j twice the size of L_{j-1}. Obviously, by this we maintain the first invariant specified above. In case the size of L_j exceeds the polynomial bound we fixed, we use repeatedly the *Trimming Rule* in order to throw candidates out of L_j until we are within the maximal size allowed. The Trimming Rule never throws away the correct candidate (i.e. $S_{\lceil \frac{n}{2} \rceil + 1, l(j)}$).

3.2.4 Keeping the size of L_j bounded

Suppose that we decide to trim L_j whenever the difference between the largest candidate in it, denoted by v^j_{max}, and the smallest candidate in it, denoted by v^j_{min}, exceeds a certain polynomial, say n^α (for some constant α). At least one of the values v^j_{max}, v^j_{min} is not the correct value $S_{\lceil \frac{n}{2} \rceil + 1, l(j)}$. Therefore the trimming rule (to be defined in the sequel) may throw one of them out of the list. For this purpose, we are going to define a new secret S', for which $g^{S'}$ can be efficiently computed given $Y = g^S$, v^j_{max} and v^j_{min}. We will examine a certain position in it (which is a function of j), henceforth referred to as the *crucial position* (and shortly denoted *cp*). Essentially, S' will have the following properties:

1. If v^j_{min} is the correct candidate (i.e. $S_{\lceil \frac{n}{2} \rceil + 1, l(j)} = v^j_{min}$) then the cp-bit in S' is 0, so are the $\lceil \alpha \log n \rceil$ bits to its right, and so are all the bits to its left.
2. If v^j_{max} is the correct candidate (i.e. $S_{\lceil \frac{n}{2} \rceil + 1, l(j)} = v^j_{max}$) then the cp-bit in S' is 1, the $\lceil \alpha \log n \rceil$ bits to its right are all 0's, and so are all the bits to its left.

Consequently, in these two situations we will be able to perform the randomization we wanted. We first shift S' to the left until the *cp*-bit is placed in the $(i+1)^{st}$ location (by repeated squaring). We then multiply the result by g^r for some randomly chosen $r \in \{0,1\}^i$. The probability to have a carry into the $(i+1)^{st}$ location from the addition of r and the shifted S', is no more than $\frac{1}{n^\alpha}$ (a carry might occur only when $r_{i, i - \lceil \alpha \log n \rceil} = 11\ldots1$). Hence, by using a polynomial number of queries to the oracle (with independently chosen r's) we are able to deduce the value of the *cp*-bit by comparing the fraction of 1-answers with β and γ.

As the value of the cp-bit is revealed, we can discard one of the candidates v^j_{min} or v^j_{max} from the list: If $cp = 1$ we are guaranteed that v^j_{min} is not correct, and if $cp = 0$ we are guaranteed that v^j_{max} is not correct.

Note that in case neither v^j_{max} nor v^j_{min} are correct, we cannot ensure that the $\lceil \alpha \log n \rceil$ bits to the right of the *cp*-bit in S' will be zeros, so a carry may reach the $(i+1)^{st}$ position. Thus we can get the frequency of 1-answers altogether different from β and γ. Yet in that case, it is ok for the trimming rule to discard either one of the extreme values from the list.

We proceed with a formal presentation of the proof.

3.2.5 Definition of S' and cp

Recall that $l(j) = \lceil \frac{n}{2} \rceil + 1 - j$. We define the new secret S' (which is a function of j, S v^j_{min} and v^j_{max}) to be

$$S' = \left\lceil \frac{2^{\lceil \alpha \log n \rceil + m}}{v^j_{max} - v^j_{min}} \right\rceil \cdot (S - v^j_{min} \cdot 2^{l(j)})$$

where m is a natural number. We will see that in the choice of m there is a tradeoff between the running time and the probability of error: When m is large, the error probability is smaller. On the other hand, when m is small, the running-time is shorter (we will see that choosing m to be $\lceil \alpha \log n \rceil$ will be adequate). Note that $g^{S'}$ can be efficiently evaluated given $Y = g^S$, v^j_{min} and v^j_{max}. The Crucial Position in S' is defined to be

$$cp = \lceil \alpha \log n \rceil + m + l(j) + 1$$

Since we decided to trim L_j whenever the difference between the extreme values in it exceeds n^α, the trimming rule will be applied only for j's greater than $\lceil \alpha \log n \rceil$ (for smaller j's v^j_{max} and v^j_{min} will not differ by more than n^α). Therefore, the maximal value for cp will be $\lceil n/2 \rceil + m + 1$. Thus, for i's smaller than $\lceil n/2 \rceil + m$ it occurs that cp is greater than $i + 1$. For these i's we have to guess the $\lceil n/2 \rceil + m - i \leq m$ most significant bits of S (in order to keep the number of guesses polynomial, we restrict m to be logarithmic in n and prefered as small as possible).

3.2.6 The actual algorithms and their analysis

We first describe the procedure "find S" that on input $N \in N_n$ and i (the index of the hybrid for which the acceptance probability of D on $H^i_{N,g}$ and $\overline{H}^i_{N,g}$ differs by more than $\frac{1}{n^c}$), finds S. We proceed with an analysis of the procedure which leads us to the exact formulation of the trimming rule.

Procedure "Find S":

On input N and i execute the following steps:

1. Let $j_0 = max(\lceil n/2 \rceil + m - i, 0))$.
   Recall that $i \geq \lceil n/2 \rceil$, therefore $j_0 \in \{0, \ldots, m\}$.
2. If $j_0 > 0$ guess the j_0 most significant bits of S, and let $w \in \{0, \ldots, 2^{j_0} - 1\}$ denote the guess (if $j_0 = 0$ let $w \stackrel{\text{def}}{=} 0$).
   For each of these polynomial number of guesses do the following stages:
3. Let $L_{j_0} = \{w\}$.
4. For $j = j_0 + 1$ to $\lceil n/2 \rceil + 1$ do the following:
   (a) Let $L_j \stackrel{def}{=} \{2u,\ 2u + 1\ : u \in L_{j-1}\}$.
   Order the resulting list from the largest element v^j_{max} to the smallest element v^j_{min}.
   (b) If $v^j_{max} - v^j_{min} > 2^{\lceil \alpha \log n \rceil}$ (we are guaranteed that $v^j_{max} - v^j_{min} \leq 2 \cdot 2^{\lceil \alpha \log n \rceil}$ by the previous stage) use the trimming rule (to be specified) repeatedly until the difference between the largest element in the list and the smallest one is no more than $2^{\lceil \alpha \log n \rceil}$.
5. Check all values $v \in L_{\frac{n}{2}+1}$ and see whether g^v equals Y. If such a value is found, then it is S.

Two facts: We turn to make two observations which lead us to the formulation of the rule by which we trim L_j (assuming that $2^{\lceil \alpha \log n \rceil} < v^j_{max} - v^j_{min} \leq 2^{\lceil \alpha \log n \rceil + 1}$):

Fact 2 *Suppose that v^j_{min} indeed equals $S_{\lceil n/2 \rceil + 1, l(j)}$. Then, the cp-bit in S' is 0, all the bits to its left are 0's, and the $\lceil \alpha \log n \rceil$ bits to its right are 0's as well.*

Fact 3 *Suppose that v^j_{max} indeed equals $S_{\lceil n/2 \rceil + 1, l(j)}$. Then, the cp-bit in S' is 1, all the bits to its left are 0's, and the $l \stackrel{def}{=} min(\lceil \alpha \log n \rceil, m-1) - 1$ bits to its right are 0's as well.*

Proof:(of Fact 2) Using $v^j_{max} - v^j_{min} > 2^{\lceil \alpha \log n \rceil}$, observe that

$$\begin{aligned} S' &= \left\lceil \frac{2^{\lceil \alpha \log n \rceil + m}}{v^j_{max} - v^j_{min}} \right\rceil \cdot \left(v^j_{min} \cdot 2^{l(j)} + S_{l(j),1} - v^j_{min} \cdot 2^{l(j)} \right) \\ &\leq 2^m \cdot S_{l(j),1} \\ &\leq 2^{m + l(j)} \\ &= 2^{cp - \lceil \alpha \log n \rceil - 1} \end{aligned}$$

□

Proof:(of Fact 3) Observe that

$$\begin{aligned} S' &= \left\lceil \frac{2^{\lceil \alpha \log n \rceil + m}}{v^j_{max} - v^j_{min}} \right\rceil \cdot \left(v^j_{max} \cdot 2^{l(j)} + S_{l(j),1} - v^j_{min} \cdot 2^{l(j)} \right) \\ &= \left\lceil \frac{2^{\lceil \alpha \log n \rceil + m}}{v^j_{max} - v^j_{min}} \right\rceil \cdot \left((v^j_{max} - v^j_{min}) \cdot 2^{l(j)} + S_{l(j),1} \right) \\ &= 2^{\lceil \alpha \log n \rceil + m + l(j)} + \delta \cdot (v^j_{max} - v^j_{min}) \cdot 2^{l(j)} + \left\lceil \frac{2^{\lceil \alpha \log n \rceil + m}}{v^j_{max} - v^j_{min}} \right\rceil \cdot S_{l(j),1} \end{aligned}$$

where $\delta = \left\lceil \frac{2^{\lceil \alpha \log n \rceil + m}}{v^j_{max} - v^j_{min}} \right\rceil - \frac{2^{\lceil \alpha \log n \rceil + m}}{v^j_{max} - v^j_{min}} \in [0, 1)$.

Let $U_1 = \delta \cdot (v^j_{max} - v^j_{min}) \cdot 2^{l(j)}$ and let $U_2 = \left\lceil \frac{2^{\lceil \alpha \log n \rceil + m}}{v^j_{max} - v^j_{min}} \right\rceil \cdot S_{l(j),1}$.
We can write S' as

$$S' = 2^{cp-1} + U_1 + U_2$$

Recall that $2^{\lceil \alpha \log n \rceil} < v^j_{max} - v^j_{min} \leq 2 \cdot 2^{\lceil \alpha \log n \rceil}$. Therefore, $U_1 \leq 2 \cdot 2^{\lceil \alpha \log n \rceil + l(j)} = 2^{cp-m}$ and $U_2 \leq 2^{m+l(j)} = 2^{cp - \lceil \alpha \log n \rceil - 1}$. Also, both $U_1, U_2 \geq 0$.
Consequently S' is of the following form:

- The *cp*-bit in S' is 1 and all the bits to its left are 0's.
- Let $l = min(\lceil \alpha \log n \rceil, m-1) - 1$. Then the l bits to the right of the *cp*-bit in S' are 0's.

□

Fact 3 implies that we must choose m to be at least $\lceil \alpha \log n \rceil$, otherwise there wouldn't be enough 0's to the right of the *cp*-bit to enable the randomization. On the other hand, the larger m

is, the more bits we have to guess in Step (1) of the procedure "Find S". We therefore set m to be $\lceil \alpha \log n \rceil$, and respectively define

$$S' = \left\lceil \frac{2^{2\lceil \alpha \log n \rceil}}{v^j_{max} - v^j_{min}} \right\rceil \cdot (S - v^j_{min} \cdot 2^{l(j)})$$

and

$$cp = 2\lceil \alpha \log n \rceil + l(j) + 1$$

We now formally state the trimming rule:

Trimming Rule:

1. Shift S' by $i + 1 - cp$ bits to the left (by computing $Y' = g^{S' \cdot 2^{i+1-cp}}$) therefore placing the crucial position in S' on location $i + 1$.
2. Pick $t(n) = n^{2c+4}$ random elements $x_1, \ldots, x_{t(n)} \in \{0,1\}^i$.
3. For each $1 \leq k \leq t(n)$ query the oracle on $Y' \cdot g^{x_k} \pmod N$ and denote by b_k its answer (i.e. $b_k = D(g^{S' \cdot 2^{i+1-cp} + x_k})$). Denote by M the mean $\frac{\sum_{k=1}^{t(n)} b_k}{t(n)}$.
4. If $M \leq (\beta + \frac{\gamma - \beta}{2})$ discard the candidate value v^j_{max} from the list L_j. Otherwise (i.e. when $M > (\beta + \frac{\gamma-\beta}{2})$) discard the candidate value v^j_{min}.

Note that the trimming rule is applied only for j's that are greater than $j_0 + \lceil \alpha \log n \rceil$. Thus, $i + 1$ is always greater or equal to cp, making Step (1) in the trimming rule well defined.

Using Chernoff bound one can show that the error probability of the trimming rule (i.e. the probability that the correct value will be discarded from the list) is exponentially small (for the exact proof see Appendix A).

Claim 3.4.1 *The Procedure "Find S" combined with the Trimming Rule above can factor integers picked randomly from N_n with probability greater than $\frac{1}{8n^c}$.*

Proof: As previously mentioned, for a pair $\langle N, g \rangle$ uniformly chosen from P_n (where N is equal to $P \cdot Q$), the following two facts hold:

1. With overwhelming probability $ord_N(g) > P + Q - 1$.
2. With probability greater than $\frac{1}{2n^c}$ the pair $\langle N, g \rangle$ belongs to the set W_n.

Therefore, given a random $N = P \cdot Q$ in N_n, we can pick g randomly in Z^*_N and with probability higher than $\frac{1}{4n^c}$ both of the above conditions hold. Hence, S will be equal to $P+Q-1$ (according to Fact 1) and the algorithm D will have advantage of at least $\frac{1}{2n^c}$ in distinguishing the distributions $H^i_{N,g}$ and $\overline{H}^i_{N,g}$ (see Equation 3). Since the probability of error by the trimming rule is exponentially small, and since the trimming rule is used polynomially many times throughout the procedure "Find S", with probability greater than $\frac{1}{8n^c}$ the value S will be found. $\square$

Note that the procedure "Find S" together with the Trimming Rule yields at most $2^{j_0} \cdot 2^{\lceil \alpha \log n \rceil} \leq 2^{2\lceil \alpha \log n \rceil} = n^{O(1)}$ possible values for S, and is therefore polynomial time. Thus Claim 3.4.1 finishes the proof of Lemma 3.3.

3.3 Proof of Main Theorem

We now go back to Theorem 3.2, and prove it using Lemma 3.3. Assume that the gap between the acceptance probability of D on the extreme hybrids $H_n^{\lceil n/2\rceil}$ and $H_n^{n+\omega(\log n)}$ is greater than $\frac{1}{n^d}$. We construct an algorithm A that factors integers uniformly distributed in N_n. On input N, algorithm A picks a random i in $\{\lceil\frac{n}{2}\rceil,\ldots,n+\omega(\log n)\}$ and runs the procedure "Find S" on (N, i). By Lemma 3.3, the probability that "Find S" indeed factors N, is greater than one eight of the gap between the acceptance probabilities of D on H_n^i and H_n^{i+1}, for a random i as above. Denote the number of hybrids, $\lfloor n/2\rfloor + \omega(\log n)$, by $m(n)$. Then, we have that for all sufficiently large n's

$$\begin{aligned}
\Pr[\text{A factors } N] &= \frac{1}{m(n)} \sum_{i=\lceil n/2\rceil}^{n+\omega(\log n)} \Pr[\text{“Find S” on input } (N,i) \text{ factors } N] \\
&\geq \frac{1}{m(n)} \sum_{i=\lceil n/2\rceil}^{n+\omega(\log n)} \frac{1}{8}\cdot \left|\Pr[D(H_n^{i+1})=1] - \Pr[D(H_n^{i})=1]\right| \\
&\geq \frac{1}{m(n)} \cdot \frac{1}{8} \cdot \left|\Pr[D(H_n^{n+\omega(\log n)})=1] - \Pr[D(H_n^{\lceil n/2\rceil})=1]\right| \\
&\geq \frac{1}{n^{d+1}}
\end{aligned}$$

thus contradicting Assumption 1.

Remark: In fact, Theorem 3.2 holds even when the distribution $Half_n$ is defined to include all triplets of the form $\langle N, g, g^x\rangle$ where $\langle N, g\rangle \in_R P_n$ and $x \in_R \{0,1\}^{\lceil n/2\rceil - O(\log n)}$ (rather than $x \in_R \{0,1\}^{\lceil n/2\rceil}$). The original proof should then be modified so that in Step (1) of procedure "Find S", the index j_0 may belong to the set $\{0,\ldots,m+O(\log n)\}$ (instead of being in $\{0,\ldots,m\}$). Thus, we need to guess more bits from S (in Step (2) of procedure "Find S" the j_0 most significant bits of S are guessed), however the total number of possible guesses remains polynomial in n.

3.4 Equivalence to the HSS result

Theorem 3.2 is actually equivalent to the result by [HSS] on the simultaneous hardness of the upper $\lceil n/2\rceil$ bits in the exponentiation function $f_{N,g}$. In order to show that, we discuss first an alternative version of Theorem 3.2. Recall the hybrid $H_n^{n+\omega(\log n)}$ defined in the proof of Theorem 3.2, including triplets $\langle N, g, g^R\rangle$, where $\langle N, g\rangle$ is uniformly distributed in P_n and R is uniformly distributed in $\{0,1\}^{n+\omega(\log n)}$. Let us denote it by $\widetilde{Full}_n$. The following is a corollary from Theorem 3.2 and from Claim 3.2.1.

Corollary 3.5 *The probability ensembles $\{Half_n\}_{n\in N}$ and $\{\widetilde{Full}_n\}_{n\in N}$ are computationally indistinguishable.*

We show that Corollary 3.5 is equivalent to the result of [HSS]. But first, let us give the exact formulation of their result.

Definition 3.6 *Let $\langle N, g\rangle$ be uniformly distributed in P_n, let x be uniformly distributed in $\{0,1\}^n$ and let r be uniformly distributed in $\{0,1\}^{\lceil\frac{n}{2}\rceil}$. We define the following probability distributions:*

$$X_n \stackrel{def}{=} \langle N, g, f_{N,g}(x), x_{n,\lceil n/2\rceil}\rangle$$

and

$$Y_n \stackrel{def}{=} \langle N, g, f_{N,g}(x), r\rangle$$

Theorem 3.7 (Håstad et al.) *The probability ensembles $\{X_n\}_{n\in N}$ and $\{Y_n\}_{n\in N}$ are computationally indistinguishable.*[7]

3.4.1 The Equivalence

Theorem 3.8 *Theorem 3.7 holds if and only if Corollary 3.5 holds.*

Proof: We show how to transform a probabilistic polynomial-time algorithm D that distinguishes the ensemble $\{X_n\}$ from $\{Y_n\}$ into a probabilistic polynomial-time algorithm D' that distinguishes the ensemble $\{Half_n\}$ from $\{\widetilde{Full}_n\}$, and vice versa.

Transforming D into D': On input $\langle N, g, y\rangle$, pick z uniformly from $\{0,1\}^{\lceil n/2\rceil}$ and run D on $\langle N, g, y \cdot g^{z\cdot 2^{\lceil n/2\rceil}}, z\rangle$. Return D's answer as output. Observe that

1. If $\langle N, g, y\rangle$ is taken from $Half_n$, then $y = g^r$ where $r \in \{0,1\}^{\lceil n/2\rceil}$. Therefore, we have that $\langle N, g, g^{z\cdot 2^{\lceil n/2\rceil}+r}, z\rangle$ is distributed as X_n.
2. If $\langle N, g, y\rangle$ is taken from $\widetilde{Full}_n$, then $y = g^R$, where $R \in \{0,1\}^{n+\omega(\log n)}$. Let $\approx$ denote statistical closeness. Note that

$$\begin{aligned}(U_{n+\omega(\log n)} + z\cdot 2^{\lceil n/2\rceil}) \bmod ord_N(g) &\approx U_{n+\omega(\log n)} \bmod ord_N(g)\\ &\approx U_n \bmod ord_N(g)\end{aligned}$$

(the proof of each of the above transitions is similar to the proof of Claim 3.2.1). Therefore, $\langle N, g, g^{z\cdot 2^{\lceil n/2\rceil}+R}, z\rangle$ is statistically close to Y_n.

Thus, Theorem 3.7 is implied by Corollary 3.5.

Transforming D' into D: On input $\langle N, g, y, z\rangle$, run D' on $\langle N, g, y/g^{z\cdot 2^{\lceil n/2\rceil}}\rangle$ and output D''s answer. Observe that

1. If $\langle N, g, y, z\rangle$ is taken from X_n, then $y = f_{N,g}(x) = g^x$ and $z = x_{n,\lceil n/2\rceil}$. Therefore, $y/g^{z\cdot 2^{\lceil n/2\rceil}} = g^{x_{\lceil n/2\rceil,1}}$ and thus $\langle N, g, y/g^{z\cdot 2^{\lceil n/2\rceil}}\rangle$ is uniformly distributed in $Half_n$.
2. If $\langle N, g, y, z\rangle$ is taken from Y_n, then $y = f_{N,g}(x) = g^x$ and z is independent of x. Note that

$$\begin{aligned}(U_n - z\cdot 2^{\lceil n/2\rceil}) \bmod ord_N(g) &= U_n \bmod ord_N(g) - z\cdot 2^{\lceil n/2\rceil} \bmod ord_N(g)\\ &\approx U_{n+\omega(\log n)} \bmod ord_N(g) - z\cdot 2^{\lceil n/2\rceil} \bmod ord_N(g)\\ &= (U_{n+\omega(\log n)} - z\cdot 2^{\lceil n/2\rceil}) \bmod ord_N(g)\\ &\approx U_{n+\omega(\log n)} \bmod ord_N(g)\end{aligned}$$

Therefore, $\langle N, g, y/g^{z\cdot 2^{\lceil n/2\rceil}}\rangle$ is statistically close to $\widetilde{Full}_n$.

Thus, Theorem 3.7 implies Corollary 3.5.

■

[7] Actually, the simultaneous hardness of the upper $\lceil \frac{n}{2}\rceil$ in $f_{N,g}$ was defined differently by [HSS]. Their definition states that the two distributions $\langle \tilde{x}_{n,\lceil n/2\rceil}, Z\rangle$ and $\langle r, Z\rangle$ are computationally indistinguishable, where $Z = g^x$ (for $x \in_R Z_N^*$), $\tilde{x} = DL_{N,g}(Z)$ and $r \in_R \{0,1\}^{\lceil n/2\rceil}$. However, this definition is problematic: At least the most significant bit in the first distribution, $\tilde{x}_n$, will be always 0, since $ord_N(g)$ is always smaller than $N/2$. Hence the above two distributions can be easily distinguished.

3.4.2 Discussion

Our proof of Theorem 3.2 simplifies to a great extent the proof given by [HSS] to Theorem 3.7. Basically, this is due to the following reasons:

1. Unlike in [HSS], we do not require that the order of g in Z_N^* will be very high (i.e. greater than $\frac{1}{n^k} \cdot (P-1)(Q-1)$). It suffices that the order of g will be greater than $P+Q-1$.
2. We do not need to consider separately the $O(\log n)$ most significant bits as done in [HSS] (where a very complex proof is given for these bits).
3. As a consequence from the different nature of the oracles, the randomization conducted by us (randomizing the bottom i bits) is different from the randomization done in [HSS] (randomizing the full range $[0, ord_N(g))$). Therefore many of the difficulties encountered in the work of [HSS] are not relevant in our proof. For example, we do not need to avoid a wrap around the order of g.

Further discussion of the above equivalence is given in [R, Sec. 3.3.3].

4 Application to Pseudorandom Generators

An immediate application of Theorem 3.2 is an efficient factoring-based pseudorandom generator which nearly doubles the length of its input. The key tool used is a construction by Goldreich and Wigderson of a tiny family of functions which has good extraction properties [GW]. We also discuss how the parameters of the generator (a composite $N \in N_n$ and an element $g \in Z_N^*$) can be chosen in a randomness-efficient way (which is polynomial-time). In particular, we present a method of choosing a random n-bit prime using only a linear number of random bits. This translates to a hitting problem which can be solved efficiently using methods described in [G2].

4.1 Our construction vs. the HSS construction

Looking at Theorem 3.2, the first construction that comes to mind is a "pseudorandom generator" that takes a seed r of length $\lceil n/2 \rceil$ and outputs $g^r \bmod N$ (for a fixed pair $\langle N, g\rangle$ in P_n). However, the output of the above so-called "pseudorandom generator" is not really pseudorandom. Even though it is computationally infeasible to distinguish between it and the distribution $g^R \bmod N$ (for a random R in $[0, ord_N(g))$), we are not guaranteed that it cannot be easily told apart from the uniform distribution on n-bit strings. The same applies for a "pseudorandom generator" implied directly by Theorem 3.7 (of [HSS]), which takes a seed x of length n, and outputs $g^x \bmod N$ followed by $x_{\lceil n/2 \rceil, 1}$ (again, for fixed $\langle N, g\rangle$ in P_n).

Denote by $Half_{N,g}$ the distribution $g^r \bmod N$, where r is uniformly distributed over strings of length $\lceil n/2 \rceil$, and by $Full_{N,g}$ the distribution $g^R \bmod N$, where R is uniformly distributed over $[0, ord_N(g))$. Observe that the "amount of randomness" that $Full_{N,g}$ encapsulates in it is high, in the sense that it does not assign a too large probability mass to any value. More formally, we measure the "amount of randomness" in terms of *min-entropy.*

Definition 4.1 *Let X be a random variable. We say that X has min-entropy k, if for every x we have that* $\Pr(X = x) \leq 2^{-k}$.

The distribution $Full_{N,g}$ has min-entropy greater than κ, where

$$\kappa \stackrel{def}{=} \kappa(N, g) \stackrel{def}{=} \lfloor \log(ord_N(g)) \rfloor$$

The following fact is an immediate consequence of Proposition 3.4:

Fact 4 *Let $\langle N, g\rangle$ be uniformly distributed in P_n, then $\kappa \leq n - \frac{1}{2}\log^2 n$ with negligible probability.*

Using hash functions which have good extracting properties, we are able to "smoothen" the distribution $Full_{N,g}$, and extract from it an almost uniform distribution over strings of length $n - \log^2 n$. To be more formal, we use a family of functions F having an extraction property, satisfying that for all but an ϵ fraction of the functions in F, a distribution over strings of length n having min-entropy $n - \frac{1}{2}\log^2 n$ is mapped to a distribution over strings of length $n - \log^2 n$ which is ϵ-close to uniform (we refer to ϵ, which is generally taken to be negligible in n, as the quality-parameter of the extraction property achieved by F). The price we pay for the use in extractors, hides in a lower expansion factor of the pseudorandom generators. Specifically, we need to use a part of the random seed in order to choose a random function in the family F we are using. Additionally, we lose a small quantity of pseudorandom bits when applying the extracting function.

Håstad et.al. [HSS] used a universal family of hash functions [CW] in their construction of a pseudorandom generator. The quality parameter achieved by this family of functions is exponentially small in n (and therefore has the best possible quality). However, a universal family of hash functions has to be large: exponential in n. Thus the number of random bits needed to generate (and represent) a function in this family is polynomial in n, resulting in a considerably large loss in the expansion factor of their generator.

Instead, we use an explicit construction due to Goldreich and Wigderson [GW] of a family of functions, which exhibits a trade-off between the size of the family and the quality parameter ϵ of the extraction property it achieves. Specifically, they demonstrate a construction of a family of functions of size $poly(n/\epsilon)$ achieving the extraction property with quality ϵ. Taking, for example, $\epsilon = n^{-\log n}$, yields a family of functions of very good quality (not exponentially small in n but still negligible in n), where each function in the family can be represented using $O(\log^2 n)$ bits.

4.1.1 The HSS construction

We present now the construction of the HSS pseudorandom generator. Even though the expansion factor of the HSS-generator can be increased using the function families of [GW], we present the original construction that uses universal hashing.

Construction 4.2 ([HSS]): *Let $H_n^{\kappa-\log^2 n}$ be a universal family of hash functions which maps n-bit strings to $(\kappa - \log^2 n)$-bit strings, and suppose that every $h \in H_n^{\kappa-\log^2 n}$ is represented using $2n$ bits. The mapping $G_{N,g}^{HSS} : \{0,1\}^{3n} \to \{0,1\}^{3.5n-O(\log^2 n)}$ is defined as follows:*[8]
Let $x \in \{0,1\}^n$ and let $h \in H_n^{\kappa-\log^2 n}$. Then,

$$G_{N,g}^{HSS}(h,x) \stackrel{def}{=} \left(h, h(g^x), x_{\lceil n/2\rceil,1}\right)$$

Note that applying the hash function causes a loss of $O(\log^2 n)$ bits in the length of the output. Therefore, the fact that $\lceil n/2\rceil$ bits are simultaneously hard in $f_{N,g}$ (and not just $O(\log^2 n)$) is essential for the construction of $G_{N,g}^{HSS}$, since the addition of the $\lceil n/2\rceil$ least significant bits to the output of the generator more than compensates for the loss of $O(\log^2 n)$ bits. Observe that the expansion factor obtained by the HSS-construction is approximately $\frac{7}{6}$ (whereas using the [GW] construction one can improve it to approximately $\frac{3}{2}$).

[8] In fact, in the HSS construction, N is restricted to be the multiplication of two *safe primes*, see [HSS].

4.1.2 Our construction

We now present our construction of a pseudorandom generator achieving an expansion factor of nearly 2. But first we give the exact formulation of the relevant result of [GW] (the construction itself is presented in Appendix C).

Theorem 4.3 (Extractors for High Min-Entropy [GW]): *Let $k < n$ and $m < n - k$ be integers, and $\epsilon > max\{2^{-(m-O(k))/O(1)}, 2^{-(n-m-O(k)/O(1))}\}$. (In particular, $m < n - O(k)$.) There exists a family of functions, each mapping $\{0,1\}^n$ to $\{0,1\}^m$, satisfying the following:*

- *each function is represented by a unique string of length $O(k + \log(\frac{1}{\epsilon}))$.*
- *there exists a logspace algorithm that, on input a description of a function f and a string x, returns $f(x)$.*
- *for every random variable $X \in \{0,1\}^n$ of min-entropy $n - k$, all but an ϵ-fraction of the functions f in the family satisfy*
$$SD(f(X), U_m) \leq \epsilon$$

In particular, taking $k = \frac{1}{2}\log^2 n$, $m = n - \log^2 n$ and $\epsilon = n^{-\log n}$, Theorem 4.3 implies the existence of a family of functions F, mapping $\{0,1\}^n$ to $\{0,1\}^m$, where each function $f \in F$ can be represented by a string of length $O(\log^2 n)$. We are now ready to exhibit our construction of a pseudorandom generator which uses the family F.

Construction 4.4 *We define the mapping $G_{N,g} : \{0,1\}^{\lceil \frac{n}{2} \rceil + O(\log^2 n)} \to \{0,1\}^n$ as follows: Let $x \in \{0,1\}^{\lceil \frac{n}{2} \rceil}$ and let $f \in F$. Then,*

$$G_{N,g}(f, x) \stackrel{def}{=} (f, f(g^x))$$

Theorem 4.5 *$G_{N,g}$ is a pseudorandom generator.*

Proof: Obviously $G_{N,g}$ is efficiently computable (since every $f \in F$ can be evaluated in polynomial time). Let F denote the random variable obtained by selecting uniformly a function f from the family F (although bearing the same name, it will be clear from the context whether we mean the random variable F or the function family F). Observe that

$$G_{N,g}(U_{\lceil \frac{n}{2} \rceil + O(\log^2 n)}) \equiv \quad (F, F(Half_{N,g}))$$
$$U_n \equiv \quad (F, U_m)$$

Consider now the hybrid $(F, F(Full_{N,g}))$. The theorem is directly implied from the following two claims:

Claim 4.5.1 *The ensembles*

$$\{(F, F(Half_{N,g}))\}_{n \in N}$$

and

$$\{(F, F(Full_{N,g}))\}_{n \in N}$$

are computationally indistinguishable.

Proof: The existence of an efficient distinguisher D between the above ensembles implies the existence of an efficient distinguisher D' between the ensembles $Half_n$ and $Full_n$: On input $\langle N, g, y\rangle$, the distinguisher D' picks an extractor f uniformly from F and outputs D's answer on input $(f, f(y))$. □

Claim 4.5.2 *The ensembles*

$$\{(F, F(Full_{N,g}))\}_{n\in N}$$

and

$$\{(F, U_m)\}_{n\in N}$$

are statistically close.

Proof: The third property of Theorem 4.3, ensures that for all but an ϵ-fraction of the functions f in F, the statistical difference between the ensembles $F(Full_{N,g})$ and U_m is bounded from above by ϵ. Thus, the statistical difference between $(F, F(Full_{N,g}))$ and (F, U_m) is no more than 2ϵ. Since ϵ was taken to be $n^{-\log n}$, we have that the ensembles $(F, F(Full_{N,g}))$ and (F, U_m) are statistically close. □

■

4.1.3 Increasing the expansion factor of the generator

The pseudorandom generator described above almost doubles the length of its input. However, such a small expansion factor has limited value in practice. Still, it is well known that even a pseudorandom generator G producing $n+1$ bits from an n-bit seed can be used in order to construct a pseudorandom generator G' having any arbitrary polynomial expansion factor (see e.g. [G, Sec. 3.3 Thm. 3.3.3]). Unfortunately, the cost of the latter transformation is rather high: Producing each bit in G''s output requires one evaluation of G. Nevertheless, since our generator $G_{N,g}$ has an expansion factor of nearly 2 to start with, we can do better than that: $G_{N,g}$ can be used to construct a generator $G'_{N,g}$ having an arbitrary polynomial expansion factor, such that for every $n/2 - O(\log^2 n)$ bits of output, one evaluation of $G_{N,g}$ is required. We remark that the issue of increasing the expansion factor of $G_{N,g}$ is relevant mostly due to the need to randomly pick the parameters N and g, which requires $O(n)$ additional random bits (as will be explained in the subsequent subsection). Our suggestion is to pick randomly N and g, set them once and for all, and construct a pseudorandom generator having a large expansion factor using this specific $G_{N,g}$. This way the cost of picking N and g becomes negligible (compared to our "profit" from the new generator).

We describe now how in general one uses a generator $G : \{0,1\}^n \to \{0,1\}^{n+l(n)}$ (for an integer function l) to construct a generator $G' : \{0,1\}^n \to \{0,1\}^{l(n)\cdot p(n)}$, for any arbitrary polynomial $p(\cdot)$.

Construction 4.6 *Let $l : N \to N$ be an integer function satisfying $l(n) > 0$ for every $n \in N$, let $p(\cdot)$ be a polynomial and let $G : \{0,1\}^n \to \{0,1\}^{n+l(n)}$ be a deterministic polynomial-time algorithm. Define $G'(s) = \tau_1 \ldots \tau_{p(n)}$, where $s_0 \stackrel{\mathrm{def}}{=} s$, the string s_i is the n-bit long suffix of $G(s_{i-1})$ and τ_i is the $l(n)$-bit long prefix of $G(s_{i-1})$, for every $1 \le i \le p(n)$ (i.e., $\tau_i s_i = G(s_{i-1})$).*

Theorem 4.7 *If G is a pseudorandom generator then so is G'.*

Theorem 4.7 is a generalization of Theorem 3.3.3 proven in [G] (regarding a generator producing $n+1$ bits from an n bit seed). Observe that for every $l(n)$ output bits of G', one evaluation of G is required. Using our generator $G_{N,g}$ as the building block, we obtain a generator $G'_{N,g}$ that expands input of size $n/2 + O(\log^2 n)$ to output of size n^c using approximately $\frac{n^c}{n/2}$ applications of $G_{N,g}$. Since evaluating $G_{N,g}$ costs approximately $\frac{n}{4}$ modular multiplications, we have that $G'_{N,g}$ can be evaluated using approximately $\frac{n^c}{2}$ modular multiplications.

4.2 An efficient choice of the parameters (N and g)

In order to use in practice the generator $G_{N,g}$ we need to generate the parameters N and g from a primary seed in an "efficient" way, where by "efficient" we mean that both the running time and the amount of randomness used should be as small as possible. The major challenge is to generate efficiently two uniformly distributed primes P and Q, in order to obtain a random $N = P \cdot Q$ in N_n. A random element g in Z_N^* can be chosen using $O(n)$ random coins by picking a random number in $\{0,1\}^{n+\log^2 n}$ and reducing it modulo N (only with negligible probability the element obtained will not be relatively prime to N). We describe now a general method by which we can pick a random n-bit prime in polynomial time, using only a linear number of random coins.

4.2.1 Picking a random n-bit prime using $O(n)$ random bits

The trivial algorithm to choose a random n-bit prime is to repeat the following two stages until a prime x is output.

1. Choose a random integer x in $\{0,1\}^n$.
2. Test whether x is a prime. If it is, stop and output x.

Since the density of primes in $\{0,1\}^n$ is approximately $\frac{1}{n}$, the expected number of times that the above loop is performed is approximately n. Even assuming that we have a deterministic primality test, the above algorithm requires an expected $O(n^2)$ random bits. We now show how to perform poly(n) dependent iterations of the loop using only $O(n)$ random bits (rather than doing $O(n)$ independent iterations using $O(n^2)$ random bits). We will use, however, a probabilistic primality tester of Bach [Bach], which is a randomness-efficient version of the Miller-Rabin [M, R2] primality tester.

Theorem 4.8 (randomness efficient primality tester [Bach]): *There exists a probabilistic polynomial time algorithm that on input P uses $|P|$ random bits so that if P is a prime then the algorithm always accepts, and otherwise (i.e. P is a composite) the algorithm accepts with probability at most $\frac{1}{\sqrt{P}}$.*

Combining the above procedures, we have

Corollary 4.9 *There exists a probabilistic polynomial-time algorithm that uses $2n$ random coins such that*

1. *with probability $\Theta(\frac{1}{n})$ outputs an n-bit prime. Furthermore, the probability to output a specific prime is 2^{-n}.*
2. *with probability $1 - \Theta(\frac{1}{n}) - \exp(-n)$ outputs a special failure sign, denoted $\perp$.*
3. *with probability at most $2^{-n/2}$ outputs a composite.*

4.2.2 A hitting problem

We refer to the algorithm guaranteed from Corollary 4.9 as a black-box. We associate every string $s \in \{0,1\}^{2n}$ with the output of the black-box given s as its random coins. Denote by W the set of strings in $\{0,1\}^{2n}$ which are associated with an n-bit prime. Corollary 4.9 implies that the density of W within $\{0,1\}^{2n}$ is $\Theta(\frac{1}{n})$. The problem of uniformly picking an n-bit prime translates to a hitting problem, where we need to find a string $s \in W$ (which is subsequently used as random input for the black-box in order to yield a prime). An additional requirement is that the distribution of primes obtained in this way will be very close to uniform. Our goal now is to find an algorithm that hits W, whose randomness complexity is linear in n. The methods we use are described in the survey of Goldreich [G2] on samplers and will be adapted to (and analyzed in) our specific setting.

A pairwise-independent hitter Our first attempt uses a pairwise independent sequence of m uniformly distributed strings in $\{0,1\}^{2n}$. Such a sequence can be generated in the following way: We associate $\{0,1\}^{2n}$ with $F \stackrel{def}{=} GF(2^{2n})$, and select independently and uniformly $s, r \in F$. We let the i'th element in the sequence be $e_i = s + i \cdot r$ (with the arithmetic of F).[9] It can be easily seen that the generated sequence is indeed pairwise-independent.

Theorem 4.10 (A pairwise-independent hitter): *Let δ be an error parameter satisfying that $1/\delta$ is polynomial in n. There exists an efficient algorithm that uses $4n$ random coins for which the following holds:*

- *The probability to output a prime is at least $1-\delta$.*
- *The probability to output a composite is at most $\exp(-n)$.*
  (With probability $\delta - \exp(-n)$ a failure sign $\perp$ is output.)
- *The probability to output a specific prime is at least 2^{-n} and at most $\frac{n}{\delta} \cdot 2^{-n}$.*

Proof: We generate $m \stackrel{def}{=} \frac{n}{\delta}$ pairwise-independent samples $e_1, \ldots, e_m$ each uniformly distributed in $\{0,1\}^{2n}$, and run the black-box using each of the e_i's as random bits. Clearly, this procedure is efficient, since m is polynomial in n. Let

$$\zeta_i \stackrel{def}{=} \begin{cases} 1 & \text{if the black-box (using } e_i \text{ as random bits) outputs a prime} \\ 0 & \text{otherwise} \end{cases}$$

Corollary 4.9 implies that the expectation of ζ_i is $\frac{1}{n}$. Using Chebishev's Inequality we have

$$\begin{aligned} \Pr\left(\sum_{i=1}^{m} \zeta_i = 0\right) &\leq \Pr\left(\left|\frac{m}{n} - \sum_{i=1}^{m} \zeta_i\right| \geq \frac{m}{n}\right) \\ &\leq \frac{m \cdot \frac{1}{n}(1-\frac{1}{n})}{(\frac{m}{n})^2} \\ &\leq \delta \end{aligned}$$

Regarding the probability to output a composite, using a union bound we get

$$\Pr[\text{a composite is output}] \quad = \quad \Pr[\exists i \text{ s.t. } e_i \text{ yields a composite}]$$

[9] Note that the number of pairwise independent strings one can generate in this way is limited to $2^{2n} - 1$.

$$\begin{aligned} &\leq \sum_{i=1}^{m} \cdot \Pr[e_i \text{ yields a composite}] \\ &\leq \frac{n}{\delta} \cdot \exp(-n) = \exp(-n) \end{aligned}$$

where the last inequality follows from the third item of Corollary 4.9 and from the fact that for every i the point e_i is uniformly distributed over $\{0,1\}^{2n}$.

As for the probability that a specific prime p is output, the first item of Corollary 4.9 implies that for every i, the probability that p is output using e_i as random coins is exactly 2^{-n} (since e_i is uniformly distributed). Thus,

$$\Pr[p \text{ is output}] \geq \Pr[e_1 \text{ yields } p] = 2^{-n}$$

On the other hand, using a union bound,

$$\Pr[p \text{ is output}] \leq \sum_{i=1}^{m} \cdot \Pr[e_i \text{ yields } p] = \frac{n}{\delta} \cdot 2^{-n}$$

■

If we were willing to settle with a polynomially small error δ (i.e., $\delta = \frac{1}{\text{poly}(n)}$) then the above algorithm would be sufficient for us. However, in order to achieve an overwhelming probability of success (i.e., $\delta = 2^{-n}$) we must take a somewhat more complex approach, which involves random walks on expander graphs (for definition and construction of expanders as well as the major theorem concerning random walks on expanders see Appendix B).

A combined hitter ¿From the pairwise independent hitter emerges another hitting problem: Let W' be the set of strings in $\{0,1\}^{4n}$, that when supplied to the pairwise-independent hitter (with a constant error parameter δ) as a random seed, makes it hit W (i.e. yield a prime). ¿From the first item of Theorem 4.10 we get that the density of W' within $\{0,1\}^{4n}$ is greater than $1-\delta$. Our new goal is to hit W' with an overwhelming probability of success.

In order to do that, we generate a random walk on an expander with vertex set $\{0,1\}^{4n}$, and use each of the vertices along the path as a seed for the pairwise-independent hitter. Taking advantage of the hitting property of expanders (see appendix B), we will have that a random walk of linear length (in n) will be sufficient in order to hit W'. Details follow.

Theorem 4.11 *There exists an efficient algorithm which uses $O(n)$ random coins such that the following holds:*

- *The probability that no prime is output is* $\exp(-n)$.
- *The probability that a composite is output is* $\exp(-n)$.
- *The probability that a specific prime is output is at least* 2^{-n} *and at most* $O(n^2) \cdot 2^{-n}$.

Proof: We use an explicit construction of expander graphs with vertex set $\{0,1\}^{4n}$, degree d and second eigenvalue λ such that $\lambda/d < 0.1$. We generate a random walk of (edge) length n on this expander using $O(n)$ random coin flips ($4n$ bits are used to generate the initial vertex and $\log d$ bits are used to obtain each additional vertex on the path). We use each of the vertices $s_1, \ldots, s_n$ along the path as random coins for the pairwise-independent hitter which makes $m = 3n$ trials (i.e., for every $1 \leq i \leq n$ we generate a pairwise-independent sequence $e_1^i, \ldots, e_m^i$ from s_i and run the black-box using each one of the e_j^i's as random bits). Recall that W' was defined to be the set of coin tosses which make the pairwise-independent hitter output a prime. ¿From Item 1 of Theorem 4.10 (with $\delta = 1/3$) we have that $|W'|/2^{4n} \geq \frac{2}{3}$. Using Theorem B.2, the probability that all vertices of a random path reside outside of W' is bounded from above by $(0.34 + 0.1)^n < 2^{-n}$. Thus,

$$\Pr[\text{no prime is output}] < 2^{-n}$$

Let us now compute the probability to output a composite.

$$\begin{aligned}
\Pr\left[\text{a composite is output}\right] &= \Pr[\exists i \text{ s.t. } s_i \text{ yields a composite}] \\
&\leq \sum_{i=1}^{n} \cdot \Pr[s_i \text{ yields a composite}] \\
&\leq n \cdot \exp(-n)
\end{aligned}$$

where the last inequality follows from the second item of Theorem 4.10 and from the fact that, for every i, the seed s_i is uniformly distributed.

In order to bound the probability that a specific prime p is output, observe that for every i and j, the point e_j^i (i.e., the j'th point in the sequence of pairwise-independent strings generated from s_i) is uniformly distributed in $\{0,1\}^n$. Thus,

$$\Pr[p \text{ is output}] \geq \Pr[e_1^1 \text{ yields } p] = 2^{-n}$$

On the other hand, applying a union bound we get

$$\Pr[p \text{ is output}] \leq \sum_{i=1}^{n} \sum_{j=1}^{m} \Pr[e_j^i \text{ yields } p] = n \cdot m \cdot 2^{-n} = 3n^2 \cdot 2^{-n}$$

■

4.2.3 Using almost uniformly distributed primes

Although the algorithm guaranteed from Theorem 4.11 does not yield uniformly distributed n-bit primes, the distribution of the primes it outputs is close to being uniform, in a sense that is quite sufficient for our needs: Denote by D_n the distribution of composites $N = P \cdot Q$ in N_n obtained by picking the primes P and Q using the algorithm of Theorem 4.11, and consider a slightly different factoring assumption, in which N is distributed according to D_n. Observe that the revised factoring assumption holds if and only if the original factoring assumption (with N uniformly distributed in N_n) holds: Let A be a probabilistic polynomial-time algorithm. Then, according to the third item of Theorem 4.11,

$$\frac{\sum_{N \in N_n} \Pr[\text{A factors } N]}{2^n} \leq \Pr[\text{A factors } N | N \sim D_n] \leq \frac{\sum_{N \in N_n} \Pr[\text{A factors } N]}{2^n/\text{poly}(n)} \tag{6}$$

where $N \sim D_n$ means that N is drawn according to the distribution D_n.
Note that the size of N_n is approximately $\frac{2^n}{n^2}$. Therefore,

$$\Pr[\text{A factors } N | N \in_R N_n] = \frac{n^2}{2^n} \sum_{N \in N_n} \Pr[\text{A factors } N] \tag{7}$$

¿From 6 and 7 we have that

$$\frac{\Pr[\text{A factors } N | N \in_R N_n]}{n^2} \leq \Pr[\text{A factors } N | N \sim D_n] \leq \frac{\Pr[\text{A factors } N | N \in_R N_n]}{n^2/\text{poly}(n)} \tag{8}$$

Thus, A does not violate the original factoring assumption if and only if it does not violate the revised factoring assumption.

Another important observation is that in all our theorems (and in particular, in Theorem 3.2), the values N and g are fixed throughout the whole proof. Thus, these theorems still hold when considering any distribution whatsoever of N (and g), provided that factoring is intractable for such a distribution.

Therefore, we have that under the standard factoring assumption (with N uniformly distributed in N_n), all our theorems hold even when the distribution of N is taken to be D_n, and the distribution of g is uniform over Z_N^*.

References

[ACGS] W. B. Alexi, B. Chor, O. Goldreich and C. P. Schnorr, RSA and Rabin functions: certain parts are as hard as the whole, *SIAM J. Comput.*, vol. 17(2), 1988, pp. 194-209.

[AKS] M. Ajtai, J. Komlos and E. Szemerédi, Deterministic simulation in LogSpace, *19th ACM Symposium on the Theory of Computing*, 1987, pp. 132-140.

[Bach] E. Bach, How to generate factored random numbers, *SIAM J. Comput.*, vol. 17(2), 1988, pp. 179-193.

[BBS] L. Blum, M. Blum and M. Shub, A Simple Secure Unpredictable Pseudo-Random Number Generator, *SIAM J. Comput.*, vol. 15, 1984, pp. 364-383.

[BM] M. Blum and S. Micali, How to generate cryptographically strong sequence of pseudo-random bits, *SIAM J. Comput.*, vol. 13, 1984, pp. 850-864.

[Chor] B. Chor, Two issues in public key cryptography: RSA bit security and a new knapsack type system, *MIT press*, 1986.

[CW] L. Carter and M. Wegman, Universal Hash Functions, *Journal of Computer and System Science*, Vol. 18, 1979, pp. 143-154.

[CoWi] A. Cohen and A. Wigderson, Dispensers, Deterministic Amplification, and Weak Random Sources, *30th FOCS*, 1989, pp. 14–19.

[G] O. Goldreich, **Foundations of Cryptography** *Fragments of a Book*, 1995. publicized at `http://www.eccc.uni-trier.de/eccc-local/ECCC-Books/eccc-books.html` (Electronic Colloquium on Computational Complexity).

[G2] O. Goldreich, A sample of samplers: A computational perspective on sampling, ECCC 4(020), 1997.

[GG] O. Gaber and Z. Galil, Explicit constructions of linear size superconcentrators, *Journal of Computer and System Science*, Vol. 22, 1981, pp.407-420.

[GILVZ] O. Goldreich, R. Impagliazzo, L.A. Levin, R. Venkatesan, and D. Zuckerman, Security Preserving Amplification of Hardness, *Proc. of the 31st IEEE Symp. on Foundation of Computer Science (FOCS)*, *31st FOCS*, pp. 318–326, 1990.

[GW] O. Goldreich and A. Wigderson, Tiny families of functions with random properties: A quality-size trade-off for hashing, *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, ACM, 1994, pp. 574-583.

[HN] J. Håstad, M. Näslund: The security of idividual RSA bits. *Proc. of IEEE Symp. on Foundations of Computer science*, 1998.

[HSS] J. Håstad, A.W. Schrift and A. Shamir, The discrete logarithm modulo a composite hides $O(n)$ bits, *J. of Computer and System Sciences*, vol. 47, 1993, pp. 376-404.

[Kah] N. Kahale, Eigenvalues and expansionsof regular graphs, *Journal of the ACM* 42(5), 1995, pp. 1091-1106.

[Kal] B. Kaliski, Jr. A pseudo-random bit generatorbased on elliptic logarithms. In A. Odlyzko, editor, *Advances in Cryptology: Proceedings of CRYPTO '86*, 1987, pp. 84-103.

[LW] D.L. Long and A. Wigderson, The discrete logarithm Hides $O(\log n)$ bits, *SIAM J. Computing*, Vol. 17, No. 2, 1988, pp. 363-372.

[M] G. L .Miller, Riemann's hypothesis and tests for priamlity, *JCSS*, Vol. 13, 1976, pp. 300-317.

[P] R. Peralta, Simultaneous security of bits in the discrete log, *Advances in Cryptology - EUROCRYPT '85 (LNCS 219)*, 1986, pp. 62-72.

[R1] M. O. Rabin, Digitalized signatures and public-key functions as intractable as factorization, Technical Report, TR-212, MIT Laboratory for Computer Science, 1979.

[R2] M. O. Rabin, Probabilistic algorithm for testing primality, *Jour. of Number Theory*, Vol. 12, 1980, pp.128-138.

[R] V. Rosen, On the security of modular exponentiation, technical report MCS00-20, Faculty of Mathematics and Computer Science, Weizmann Institute of Science, Rehovot, Israel, 2000.

[VV] U. V. Vazirani and V. V. Vazirani, Efficient and secure pseudo-random number generators, *Proceeding of 25'th FOCS*, 1984, pp.458-463.

Appendix A: Exact Analysis of Theorem 1

We show that the probability of error by the trimming rule is exponentially small. Suppose we want to trim the list L_j and that v^j_{min} is the correct value $S_{\lceil \frac{n}{2} \rceil+1,l(j)}$ (the analysis in the case where v^j_{max} is the correct value is analogous). Let δ denote S' shifted by $i+1-cp$ positions to the left, that is, let $\delta = S' \cdot 2^{i+1-cp}$. By Fact 2 we have that $\delta \leq 2^{i-\lceil \alpha \log n \rceil}$ (since $S' \leq 2^{cp-\lceil \alpha \log n \rceil -1}$). Recall that b_k is the oracle answer on the query $g^{\delta+x_k}$ (see Step (3) of the trimming rule). Let us bound the expectation of b_k:

$$\begin{aligned}
E(b_k) = {} & \Pr\left[D(g^{x_k+\delta}) = 1 | 0 \leq x_k \leq 2^i - 1\right] \\
= {} & \Pr\left[D(g^{x_k+\delta}) = 1 | 0 \leq x_k \leq 2^i - 1 - \delta\right] \cdot \Pr\left[0 \leq x_k \leq 2^i - 1 - \delta\right] \quad + \\
& \Pr\left[D(g^{x_k+\delta}) = 1 | 2^i - 1 - \delta < x_k \leq 2^i - 1\right] \cdot \Pr\left[2^i - 1 - \delta < x_k \leq 2^i - 1\right]
\end{aligned}$$

Let $y_k = x_k + \delta$. Then,

$$\begin{aligned}
\Pr\left[D(g^{x_k+\delta}) = 1 | 0 \leq x_k \leq 2^i - 1 - \delta\right] & = \Pr\left[D(g^{y_k}) = 1 | \delta \leq y_k \leq 2^i - 1\right] \\
& \leq \tfrac{2^i}{2^i-\delta} \cdot \Pr\left[D(g^{y_k}) = 1 | 0 \leq y_k \leq 2^i - 1\right] = \tfrac{\beta \cdot 2^i}{2^i - \delta}
\end{aligned}$$

and therefore

$$E(b_k) \leq \tfrac{\beta \cdot 2^i}{2^i-\delta} \cdot \tfrac{2^i-\delta}{2^i} + 1 \cdot \tfrac{\delta}{2^i} = \beta + \tfrac{\delta}{2^i}$$

A standard application of Chernoff bound yields:

$$\begin{aligned}
\Pr\left[\text{discard } v^j_{min}\right] & = \Pr\left[\textstyle\sum_{k=1}^t b_k > (\beta + \frac{\gamma-\beta}{2}) \cdot t\right] \\
& \leq \Pr\left[|\textstyle\sum b_k - E(\sum b_k)| > (\beta + \frac{\gamma-\beta}{2}) \cdot t - E(\sum b_k)\right] \\
& = \Pr\left[|\textstyle\sum b_k - E(\sum b_k)| > \lambda \cdot E(\sum b_k)\right]
\end{aligned}$$

for $\lambda = \frac{(\beta+\frac{\gamma-\beta}{2})t - E(\sum b_k)}{E(\sum b_k)}$.

Since

$$\begin{aligned}
\frac{\lambda^2 E(\sum b_k)}{6} & = \frac{\left[(\beta+\frac{\gamma-\beta}{2})t - E(\sum b_k)\right]^2}{6 \cdot E(\sum b_k)} \geq \frac{\left[(\beta+\frac{\gamma-\beta}{2})t - (\beta + \frac{\delta}{2^i})t\right]^2}{6 \cdot (\beta + \frac{\delta}{2^i})t} \\
& = \frac{(\frac{\gamma-\beta}{2} - \frac{\delta}{2^i})^2}{6(\beta+\frac{\delta}{2^i})} \cdot t \geq \frac{(\frac{1}{2n^c} - \frac{1}{n^\alpha})^2}{6(\beta + \frac{1}{n^\alpha})} \cdot t \geq n
\end{aligned}$$

for $\alpha = c+1$ and for $t \geq n^{2c+4}$.

Therefore, the probability of discarding v^j_{min} from the list L_j is smaller than 2^{-n}. As mentioned above, a similar argument holds for the second case, where the correct candidate is v^j_{max}. Since for every $j_0 \leq j \leq n/2+1$ we use repeatedly the trimming rule for no more than n^α times, the overall probability of error is exponentially small.

Appendix B: Expanders and Random Walks

We now define expander graphs and families of expander graphs and describe an explicit construction of expanders due to Gabber and Galil [GG]. We also state the major theorem concerning random walks on expanders. Our exposition follows that of Goldreich in [G2].

B.1 Expanders

An (N,d,λ)-expander is a d-regular graph with N vertices so that the absolute value of all eigenvalues (except the biggest one) of its adjacency matrix is bounded by λ. A (d,λ)-family is an infinite sequence of graphs so that the n^{th} graph is a $(2^n,d,\lambda)$-expander. We are interested in explicit constructions of such families of graphs, which are *efficiently constructible*, by which we mean that there exists a polynomial-time algorithm that on input n (in binary), a vertex v and an index $i \in \{1,\ldots,d\}$, returns the i'th neighbor of v.

Gaber and Galil presented such a construction of a (d,λ)-family of expanders, for $d=8$ and for some $\lambda < 8$ [GG]. Their expanders, however, are defined only for graph sizes which are perfect squares (i.e., only for even n's).

Construction B.1 [Gaber-Galil] *Let $n = 2m$. The graph G_n is defined as follows: The vertex set includes all pairs in $Z_m \times Z_m$, and each node (x,y) is connected to the four nodes $(x+y,y)$, $(x+y+1,y)$, $(x,x+y)$ and $(x,x+y+1)$.*

In our applications we use (parameterized) expanders satisfying $\frac{\lambda}{d} < \alpha$ and $d = \text{poly}(1/\alpha)$, where α is an application-specific parameter. Such (parameterized) expanders are also efficiently constructible. For example, we may obtain them by taking paths of length $O(\log 1/\alpha)$ on an expander as in construction B.1. Specifically, given a parameter $\alpha > 0$, we obtain an efficiently constructible (D,Λ)-family satisfying $\frac{\Lambda}{D} < \alpha$ and $D = \text{poly}(1/\alpha)$ as follows. We start with a constructible $(8,\lambda)$-family, set $k \stackrel{\text{def}}{=} \log_{8/\lambda}(1/\alpha) = O(\log 1/\alpha)$ and consider the paths of length k in each graph. This yields a constructible $(8^k,\lambda^k)$-family, and both $\frac{\lambda^k}{8^k} < \alpha$ and $8^k = \text{poly}(1/\alpha)$ indeed hold.

B.2 Random walks on Expanders

A fundamental discovery of Ajtai, Komlos, and Szemerédi [AKS] is that random walks on expander graphs provide a good approximation to repeated independent attempts to hit any arbitrary fixed subset of sufficient density (within the vertex set). The importance of this discovery stems from the fact that a random walk on an expander can be generated using much fewer random coins than required for generating independent samples in the vertex set. Precise formulations of the above discovery were given in [AKS, CoWi, GILVZ] culminating in Kahale's optimal analysis [Kah, Sec. 6].

Theorem B.2 (Expander Random Walk Theorem [Kah, Cor. 6.1]): *Let $G = (V,E)$ be an expander graph of degree d and λ be an upper bound on the absolute value of all eigenvalues, save the biggest one, of the adjacency matrix of the graph. Let V' be a subset of V and $\rho \stackrel{\text{def}}{=} |V'|/|V|$. Then the fraction of random walks* (in G) *of* (edge) *length ℓ which stay within V' is at most*

$$\rho \cdot \left(\rho + (1-\rho)\cdot\frac{\lambda}{d}\right)^{\ell}$$

Appendix C: Tiny Families of Functions

We now present the explicit construction of Goldreich and Wigderson of tiny families of functions designed for random variables with high min-entropy. Our exposition is taken from [GW].

We describe a construction of a family of functions, each mapping $\{0,1\}^n$ to $\{0,1\}^m$, such that all but an ϵ-fraction of them, map random-variables having min-entropy $n-k$ to a distribution whose distance from the uniform distribution is bounded by ϵ.

The construction uses an efficiently constructible expander graph, G, of degree d (power of two), second eigenvalue λ, and vertex set $\{0,1\}^m$, so that $\frac{\lambda}{d} \leq \frac{\epsilon^2}{4 \cdot 2^{k/2}}$ (and $d = \text{poly}(2^k/\epsilon)$). For every $i \in [d] \stackrel{\text{def}}{=} \{1, 2..., d\}$ and $v \in \{0,1\}^m$, denote by $g_i(v)$ the vertex reached by moving along the i^{th} edge of the vertex v. The construction uses as well a universal hashing family, denoted H, that contains hash functions each mapping $(n-m)$-bit long strings to $[d]$.

Construction C.1 *The family of functions, denoted F, is as follows: For each hashing function $h \in H$, we introduce a function $f \in F$ defined by*

$$f(x) \stackrel{def}{=} g_{h(\text{lsb}(x))}(\text{msb}(x))$$

where $\text{lsb}(x)$ *returns the $n-m$ least significant bits of $x \in \{0,1\}^n$, and* $\text{msb}(x)$ *returns the m most significant bits of x.*
Namely, $f(x)$ is the vertex reached from the vertex $v \stackrel{\text{def}}{=} \text{msb}(x)$ by following the i^{th} edge of v, where i is the image of the $n-m$ least significant bits of x under the function h.

As proven in [GW], Construction C.1 above satisfies the requirements of Theorem 4.3 (stated in Section 4).